Evolving Robot Controllers for Structured Environments Through Environment Decomposition

Rodrigo Moreno[1(✉)], Andres Faiña[2], and Kasper Støy[2]

[1] Universidad Nacional de Colombia, Bogota, Colombia
rmorenoga@unal.edu.co
[2] IT University of Copenhagen, Copenhagen, Denmark
{anfv,ksty}@itu.dk

Abstract. In this paper we aim to develop a controller that allows a robot to traverse an structured environment. The approach we use is to decompose the environment into simple sub-environments that we use as basis for evolving the controller. Specifically, we decompose a narrow corridor environment into four different sub-environments and evolve controllers that generalize to traverse two larger environments composed of the sub-environments. We also study two strategies for presenting the sub-environments to the evolutionary algorithm: all sub-environments at the same time and in sequence. Results show that by using a sequence the evolutionary algorithm can find a controller that performs well in all sub-environments more consistently than when presenting all sub-environments together. We conclude that environment decomposition is an useful approach for evolving controllers for structured environments and that the order in which the decomposed sub-environments are presented in sequence impacts the performance of the evolutionary algorithm.

Keywords: Evolutionary robotics · Environment decomposition · Sequential evolution

1 Introduction

In this paper we demonstrate how to evolve one robot controller that is able to traverse a structured environment. The approach consists of decomposing a structured environment, such as an office building or a pipe system, into simpler ones, like turns or doorways, and evolve the robot to traverse these sub-environments. A robot that can perform well in these sub-environments can generalize its behavior to any larger environment composed of the simpler ones.

The robot used is a sensor-less snake-like robot built using a simple 1 degree of freedom modular robot. There are two general controller types a robot can use, hierarchical controllers, and monolithic controllers. Using a hierarchical controller different individual controllers could be used for different sub-environments

A.M. Mora and G. Squillero (Eds.): EvoApplications 2015, LNCS 9028, pp. 795–806, 2015.
DOI: 10.1007/978-3-319-16549-3_64

requiring a decision mechanism to decide between all of them. A decision mechanism cannot be implemented in our current robot due to the lack of sensors, so we make use of a monolithic controller instead. The controller used is based on central pattern generators. Central pattern generators are commonly used to control the locomotion of complex robots that have limited computational capabilities [1].

Using a monolithic controller we have to ensure that it works in all sub-environments. Two methods can be used for this purpose: One is to evaluate the fitness of all individuals in all sub-environments at the same time. In this case performance in all sub-environments should be a compound measure of the fitness. Another approach is to introduce the sub-environments in sequence to the evolutionary algorithm. One individual is evaluated in the next sub-environment only if it performs well in the last one.

In our work a narrow corridor environment is decomposed into turns and bumps. Using the approaches of evaluating individuals in all sub-environments at the same time and in sequence we evolve controllers for these sub-environments and compare how fast and consistently a solution can be found. The best controller obtained by the evolutionary process is tested in two environments composed of the sub-environments and it is shown that it can successfully traverse through both of them. Results also show that presenting the sub-environments in a sequence improves how consistently the evolutionary process can find a solution compared to when there is no sequence present and that the order of the sequence impacts how fast a controller can be found. Overall, environment decomposition is shown to be an an useful approach for evolving controllers for structured environments.

2 Related Work

Previous task decomposition work focuses on generating behaviors for specific tasks that are combined to solve a main task [2–7]. In [2] Lee et al. use task decomposition to evolve controllers for pushing a box to a goal with a Khepera robot by evolving controllers for the sub tasks of getting to the box, circling the box and pushing the box in a certain direction. Controllers are evolved independently for each task and a decision mechanism is needed for all the different evolved controllers to work as one [3,4]. Alternatively to task decomposition, in our work instead of decomposing a given task we are given an environment that we decompose into sub-environments that could also be seen as sub-tasks.

Also, in task decomposition, it has been shown that introducing a sequence when learning multiple tasks improves the speed and reliability of the evolutionary algorithm [5,8–10]. Layered learning and incremental learning introduce the idea of a sequence in how the robot controllers are evolved for different tasks. In layered learning decomposed tasks are solved by evolving the simplest tasks separately first and then stopping the process, freezing the found solutions and using them to start new evolution processes to solve the next more complex ones. The sequence continues until the goal task has been solved. In [6] Stone et al.

evolve a controller for three tasks in a robot soccer environment: ball interception, pass evaluation and pass selection. Co-evolution is also used along layered learning to generate controllers for subtasks, in [3] Whiteson et al. co-evolve different layers at the same time as well as the decision mechanism. As it uses decision mechanisms, layered learning is a case of evolving hierarchical controllers. Here we focus on a sensor-less robot due to the ease of hardware implementation so any sensor-based decision mechanism is not possible to implement. Instead we are using a monolithic controller for it to be good in all sub-environments at the same time, something similar to what is done in incremental evolution.

In incremental evolution [7] the same task is presented to the evolutionary algorithm with various levels of difficulty starting by the easiest one. As the robot is able to solve the task the difficulty is risen gradually until the robot learns the behaviors needed to solve a desired level of complexity in the given task. The same controller is expected to include the new found behaviors without using extra parts or modules. The changes involve gradually moving the position of a goal to force the appearance of different behaviors [8], changing the height of a wall that a robot has to go over, as in [11,12], or increasing the sharpness of a curve in a maze for a mobile robot to learn how to turn [13]. Although incremental evolution involves changing the environment it remains relatively similar through the changes. Bongard et al. [8–10] show that the order in which different behaviors are incrementally learned is important for the success rate of the evolutionary process. In their work a quadruped robot with grasping capabilities is more successful in learning how to manipulate an object first and then move towards it than the other way around. Similarly, one could specify the ordering of the sub-environments in the sequence presented to the evolutionary algorithm to also change the performance of an environment decomposition learning process. Multi-objective optimization has also been used in incremental evolution to evolve behaviors without specifying a sequence [14].

The main contributions of this paper are: the idea of evolving a monolithic controller for structured environments by using environment decomposition as an alternative to task decomposition. Additionally we investigate whether introducing a sequence in how the sub-environments are presented could make the evolutionary process find a solution faster and more consistently and if the order of the sequence impacts the performance of the algorithm.

3 Experimental Setup

3.1 Simulated Modular Robot and Environments

We have modeled a snake-like robot in the V-REP simulator. V-REP is an open source robot simulator that can work with different physics engines [15]. We create a chain of 8 modules using the simple cubic modular robot shown in Fig. 1 that has only one rotational degree of freedom. Modular reconfigurable robots are a special class of robots that are built from basic units, called modules, with or without autonomy, that can reconfigure themselves to perform different

Table 1. Parameters of the simulation

Parameter	Value
Physics Engine	Bullet
Module mass	0.14 (Kg)
Max. Joint Torque	2.5 (Nm)
Dimensions	$10 \times 10 \times 15$ (cm)
Physics Time Step	0.0135 (s)

Table 2. Parameters of the controller

Parameter	Value	Range
a_r	50	-
a_x	30	-
ω	$2\pi \times 0.65$	-
w_{ij}	7	-
R	-	$[-1; 1]$
X	-	$[-1; 1]$
$\Delta\phi$	-	$[-\pi/2; \pi/2]$

tasks [16]. The module model used here is designed to be easily implemented and a first prototype has already been designed and built [17].

Each module can be connected to other modules using two connection surfaces. The module can be oriented in two ways: with it's rotational axis parallel to the horizontal plane or with the rotational axis parallel to the vertical plane. Using modules in each orientation different types of chains can be generated. In this paper a fixed chain was built with alternating orientation. The physical parameters used for the simulation can be seen in Table 1.

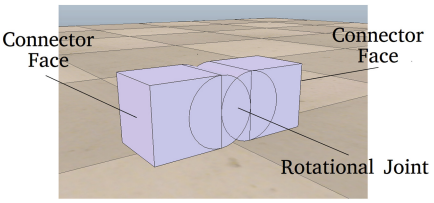


Fig. 1. The basic module is made of two cubic parts linked by a rotational joint. Each part has only one connector face.

Each module is controlled using an internal central pattern generator(CPG). Central pattern generators are neural structures that can be found in the spine of animals and that can generate complex movements from basic input. The CPGs inside the modules are modeled as phase coupled nonlinear oscillators as in [1] which provide a sinusoidal output that can be controlled by using three main parameters: Amplitude R, offset X and phase difference with neighbors $\Delta\phi$, providing a simple way of achieving coordinated movement from distributed controllers.

Equation (1) shows the coupling between different oscillator phases as a weighted sum, Eqs. (2) and (3) describe control laws that make the amplitude r and offset x in the output (4) converge to the desired values R and X. The parameters a_x, a_r and w_i are weights used to control the speed of convergence of the amplitude and offset to their respective set points and the coupling strength of the phase difference, their values can be seen on Table 2. The output θ in Eq. (4) controls the movement of the rotational joint in each module. For this experiment the same value of amplitude, offset and phase difference is used for

all modules and will be changed by the evolutionary algorithm. The values each parameter can take can be seen in Table 2.

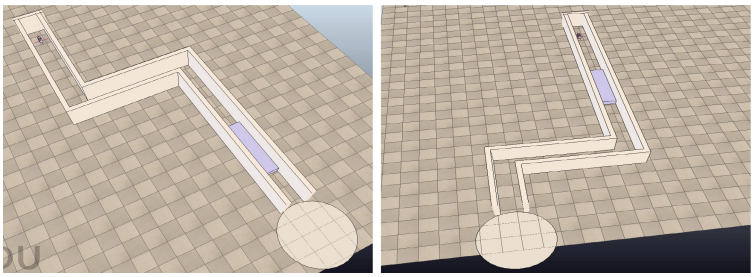
$$\dot{\phi}_i = \omega_i + \sum_j (w_{ij} Sin(\phi_j - \phi_i - \Delta\phi_{ij})) \tag{1}$$

$$\ddot{r}_i = a_r(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i) \tag{2}$$

$$\ddot{x}_i = a_x(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i) \tag{3}$$

$$\theta_i = x_i + r_i Cos(\phi_i) \tag{4}$$

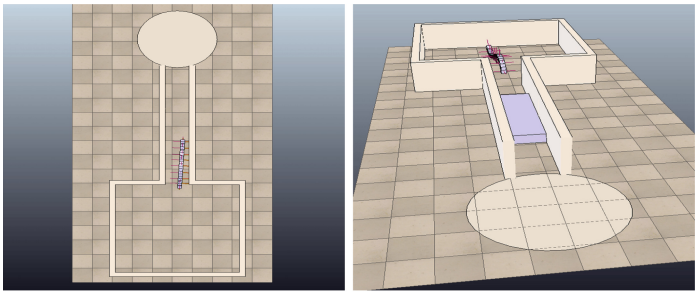
The narrow corridor environment shown in Fig. 2, in which the robot can find turns and obstacles as well as long empty segments is decomposed into four sub-environments (Fig. 3): *Straight*, *Turnleft*, *TurnRight* and *Bump*. By being good in all four sub-environments the robot should be able to move through any narrow corridor composed of these parts. The environments have been built using walls and obstacles available in the simulator.



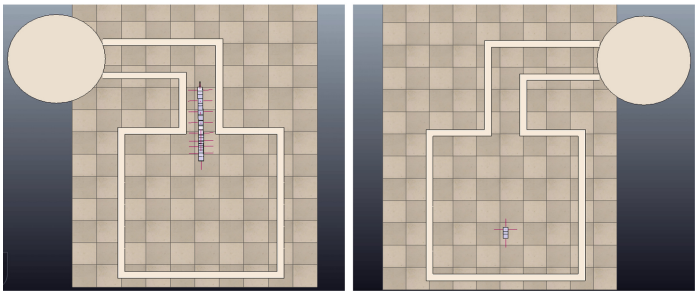
(a) (b)

Fig. 2. Two narrow corridors with turns and obstacles. The light blue rectangle is a small bump. The circle represents the goal area (Colour figure online).

The starting position of the last module of the chain in all four sub-environments can be seen in Fig. 3d. Sub-environment *Straight* (Fig. 3a) is a straight corridor in front of the starting position, *TurnLeft* (Fig. 3c) its a left turn after a shorter straight corridor, *Bump* (Fig. 3b) has a step that doubles the robot's height after some distance from the start of the same straight corridor as in *Straight*, and *TurnRight* (Fig. 3d) is a turn in the opposite direction of *TurnLeft*. The robot should move from its starting position to the end of each corridor in a limited amount of time T, and all sub-environments have a similar distance from the initial position of the robot to the goal position (circle at the end of the corridor).



(a) *Straight* (b) *Bump*



(c) *TurnLeft* (d) *TurnRight*

Fig. 3. The four sub-environments in which the robot is evaluated. The circle represents the goal area, fitness is measured as the distance to the goal along the corridor in each case. The initial position of the last module of the robot is shown in (d).

3.2 Evolutionary Algorithm

To evolve the 3 controller parameters of the CPG, namely amplitude R, offset X and phase difference with neighbors $\Delta\phi$, for the robot to get out of different sub-environments we used the Differential Evolution Algorithm [18], the specific parameters used for the algorithm are displayed in Table 3. The algorithm was implemented using the JEAF [19] framework on a 32 core AMD Opteron Linux machine. The fitness function for each sub-environment is defined in two stages: first, if the robot is not able to get out of the corridor under the maximum amount of time the fitness will be the distance to the goal D plus the maximum time allowed for the trial T in simulation time. Once the robot gets to the goal circle (Fig. 3) its fitness will be the time it takes to complete the maze t, so the fitness for each sub-environment looks like this:

$$F = \begin{cases} D + T & \text{if goal not reached} \\ t & \text{if goal reached} \end{cases} \tag{5}$$

In this way the robot can reach the goal and continue to improve its fitness by being quicker. The robot controller is evolved in two ways:

Table 3. Differential Evolution Parameters

Parameter	Value
Population Size	32
Number of Generations	300
F	0.9
CR	0.9
Max. Evaluation time T	40 (s)

Learning All Sub-environments at once. For this scenario the controller is evaluated in all four sub-environments and the total fitness is measured using two methods: the average fitness and the worst fitness of all four. In the case of the worst fitness the robot has to have a good performance in all sub-environments in order to have a good fitness. An individual has reached the goal in all sub-environments if its fitness is less than T, which is the same for all four sub-environments.

Learning All Sub-environments in Sequence. The robot is evaluated in all four sub-environments in a sequential fashion. Only if the individual being evaluated is able to get to the goal of one sub-environment under the maximum allowed time it is evaluated in the next sub-environment until an individual is capable of getting out of all four sub-environments. Sub-environments are shown to the robot in three different experiments: *Straight - TurnLeft - Bump - TurnRight* (S1), *Straight - Bump - TurnLeft - TurnRight* (S2) and *Straight - TurnRight - TurnLeft - Bump* (S3). These first three sequences cover all permutations of the sub-environments *TurnLeft*, *TurnRight* and *Bump* that are no mirrors of each other, that is turning left and then turning right is considered to be the same as turning right and then turning left. The *Straight* sub-environment is always shown to the robot at the beginning of these three initial sequences as it is the simplest and all the others include a straight element in the beginning.

The last sequence considered (*Bump - TurnRight - TurnLeft - Straight*, S4) changes this as it puts the *Straight* sub-environment at the end. Each individual receives a bonus fitness corresponding to a value that is designed to be at least greater than the maximum observed fitness a robot can get in an individual sub-environment (this parameter is based in the observed fitness of several runs of the evolution process) so the total fitness is:

$$F = \begin{cases} 1000/f_1 & \text{if goal not reached in env 1} \\ 1000/f_2 + 100 & \text{if goal reached in env 1} \\ 1000/f_3 + 200 & \text{if goal reached in env 1 and env 2} \\ 1000/f_4 + 300 & \text{if goal reached in env 1 and env 2 and env 3} \end{cases} \tag{6}$$

Being f_i the fitness obtained on sub-environment i using (5). In this case an individual has reached the goal on all sub-environments if its fitness is above

$1000/T + 300$, in this case 325. In all cases the evolution process is given a maximum of 300 generations to find a suitable controller for all four sub-environments.

4 Results

Figure 4a shows the average of the best individual fitness per generation for 10 runs of the evolutionary process using the average fitness with no sequence. Figure 4b shows the fitness for each sub-environment in one run in the case of using the average fitness. It should be noted that using the average fitness can be deceiving in that a solution can perform really well in some sub-environments while performing poorly in the others. Using the average fitness doesn't ensure that a controller is good in all sub-environments at the same time, however in this case all runs generate controllers that are successful in all sub-environments.

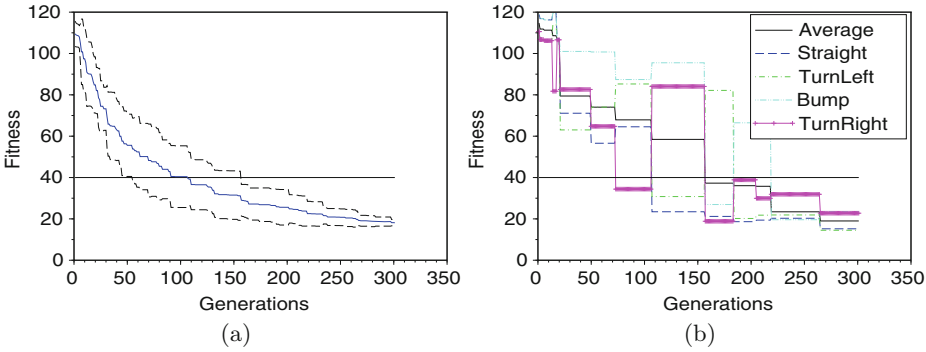


Fig. 4. Average and standard deviation of the best individual fitness per generation (a) for 10 runs evaluating all four sub-environments at the same time with the average fitness. Also, fitness in each sub-environment for 1 run (b). A fitness under 40 (*bottom line*) means an individual is successful in a sub-environment.

Figure 5 shows the best individual fitness per generation and the average of the best individual fitness per generation for 11 evolution runs using the worst fitness with no sequence. It can be seen that by using the worst fitness the evolutionary process cannot find a solution in 2 of the 11 runs for the allowed number of generations.

In Fig. 6 the average best fitness per generation for all the sequences used in the sequence learning scenario is shown. It can be seen that in all the cases where a sequence is introduced the evolutionary process is able to find a solution for all four sub-environments every time. An analysis of variance to compare the number of generations it takes for all strategies to generate a controller that is successful in all environments showed a significant difference, $F(5,53) = 3.73$, $p = 0.0057$. The means and standard deviations are presented in Table 4. Post-hoc comparisons using a Tukey HSD test showed that sequences S1 and S2 are significantly faster than S4 in generating controllers.

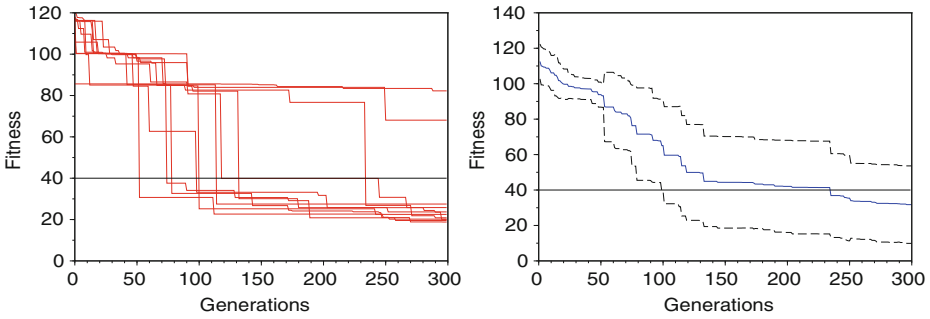


Fig. 5. Best individual fitness per generation (a) and average of best individual fitness per generation with standard deviation (b) for 11 runs of evolution evaluating all four sub-environments at the same time with the worst fitness. A fitness under 40 (*bottom line*) means an individual is successful in all sub-environments.

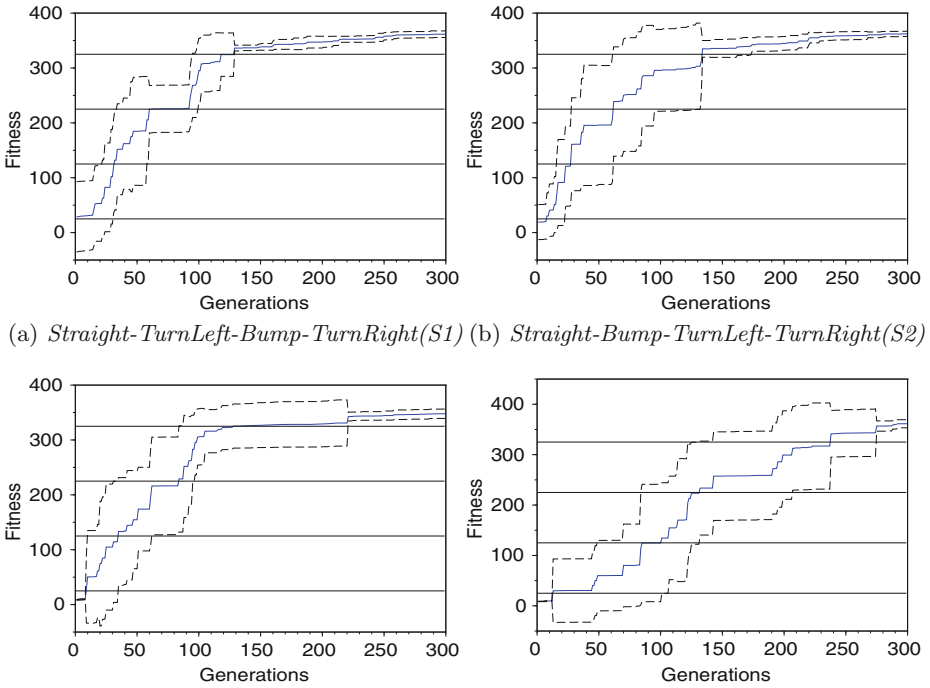


Fig. 6. Average best individual fitness per generation, with standard deviation, for 10 runs of evolution evaluating sub-environments in sequence. (*Black lines*) indicate environment transitions when a controller has successfully reached the goal in each one (Colour figure online).

Table 4. Average number of generations with standard deviation to find a controller that solves all sub-environments for all runs for all strategies.

	Worst	Average	S1	S2	S3	S4
Average	111.1	108.4	99.8	94.7	103.9	175.8
SD	52.26	51.89	22.35	48.71	52.02	59.65

When presented with the two environments from Fig. 2 the sequence S1 produces controllers that are able to traverse both corridors under 300 seconds in 18 out of 20 runs. Controllers obtained using the Average fitness are able to traverse both corridors in 14 of 20 runs and using the worst fitness in 17 of 20 runs. In contrast the sequence S4 produces controllers successful in 11 of 20 runs. Unsuccessful controllers in the larger environments can be attributed to the controllers exploiting features of the simulation to get a good fitness [20].

5 Discussion

Results show that controllers evolved by using environment decomposition were able to generalize for the larger environments even when no assumptions were made on how the robot could go from one sub-environment to another.

It can be seen that, when presenting the evolutionary algorithm with all sub-environments at the same time, although the average fitness can be a deceiving measure in this case it performs better than the worst fitness measure in evolving controllers for all sub-environments in all runs. This may be due to the worst measure giving bad fitness to controllers that perform well in almost all sub-environments but perform poorly in one. This condition is relaxed with the average fitness measure by which this kind of controllers get a better fitness.

When the sub-environments are presented in sequence it is shown that the evolutionary algorithm finds solutions for all runs, again in contrast with the worst measure case. Also, introducing a sequence ensures that a controller is good in all environments as opposed to using the average measure. The S1 sequence (Fig. 6a) performs specially well as is not only able to find solutions quickly for all four sub-environments but also in a consistent way when compared to the other strategies.

The significant difference between sequences S1,S2 and S4 indicates that the order of the sub-environments influences the result. Changing the place of the *Straight* sub-environment in the sequence (S4) makes the overall process take on average more generations to find a solution that satisfies all four sub-environments (Table 4). This indicates the idea that some environments are more or less complex to learn after learning others.

6 Conclusions and Future Work

It can be concluded that environment decomposition is an useful approach for evolving controllers for structured environments as controllers evolved in the

decomposed sub-environments generalize their behavior to more complex environments composed of the simpler ones. Introducing a sequence when presenting an evolutionary algorithm with the different sub-environments helps generate controllers more reliably and the specific sequence has an impact in the performance of the process. Future work includes investigating how the approach scales to structured, but more complex environments, using sensors for implementing a decision mechanism in the controller and considering the problem of how a robot can go from one sub-environment to another. We also aim to verify our results on the physical system once it is completed.

Acknowledgments. This project is supported in part by grant 23418 of the program "Programa nacional de proyectos para el fortalecimiento de la investigación, la creación y la innovación en posgrados en la Universidad Nacional de Colombia 2013" of Universidad Nacional de Colombia.

References

1. Crespi, A., Lachat, D., Pasquier, A., Ijspeert, A.J.: Controlling swimming and crawling in a fish robot using a central pattern generator. Auton. Robots **25**(1–2), 3–13 (2008)
2. Lee, W.P., Hallam, J., Lund, H.H.: Learning complex robot behaviours by evolutionary computing with task decomposition. In: Birk, A., Demiris, J. (eds.) Learning Robots. LNCS, vol. 1545, pp. 155–172. Springer, Heidelberg (1998)
3. Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P.: Evolving soccer keepaway players through task decomposition. Mach. Learn. **59**(1–2), 5–30 (2005)
4. Lessin, D., Fussell, D., Miikkulainen, R.: Open-ended behavioral complexity for evolved virtual creatures. In: Proceedings of the GECCO 2013, p. 335. ACM Press, New York, USA (2013)
5. Rossi, C., Eiben, A.E.: Simultaneous versus incremental learning of multiple skills by modular robots. Evol. Intell. **7**(2), 119–131 (2014)
6. Stone, P., Veloso, M.M.: Layered learning. In: de Mantaras, R.L., Plaza, E. (eds.) ECML 2000. LNCS (LNAI), vol. 1810, pp. 369–381. Springer, Heidelberg (2000)
7. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. Adapt. Behav. **5**(3–4), 317–342 (1997)
8. Bongard, J.: Behavior chaining: incremental behavioral integration for evolutionary robotics. Artif. Life XI Number **1976**, 64–71 (2008)
9. Auerbach, J., Bongard, J.C.: How robot morphology and training order affect the learning of multiple behaviors. In: IEEE Congress on CEC 2009, pp. 39–46, Trondheim, May 2009
10. Bongard, J.C.: Morphological and environmental scaffolding synergize when evolving robot controllers. In: GECCO 2011 1st workshop on evolutionary computation for designing generic algorithms, p. 179. ACM Press, Dublin, Ireland (2011)
11. Mukosaka, N., Tanev, I., Shimohara, K.: Performance of incremental genetic programming on adaptability of snake-like Robot. IES2013 **24**, 152–157 (2013)
12. Kuyucu, T., Tanev, I., Shimohara, K.: Genetic transposition inspired incremental genetic programming for efficient coevolution of locomotion and sensing of simulated snake-like robot. In: Proceedings of the Eleventh European Conference on the Synthesis and Simulation of Living Systems ECAL-2011, pp. 439–446. MIT Press, Paris (2011)

13. Song, G.B., Cho, S.B.: Combining incrementally evolved neural networks based on cellular automata for complex adaptive behaviors. In: First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, pp. 121–129. IEEE, San Antonio, TX (2000)
14. Mouret, J.-B., Doncieux, S.: Incremental Evolution of animats' behaviors as a multi-objective optimization. In: Asada, M., Hallam, J.C.T., Meyer, J.-A., Tani, J. (eds.) SAB 2008. LNCS (LNAI), vol. 5040, pp. 210–219. Springer, Heidelberg (2008)
15. Rohmer, E., Singh, S.P.N., Freese, M.: V-REP: A versatile and scalable robot simulation framework. In: 26th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), pp. 1321–1326. IEEE, Tokyo, November 2013
16. Jantapremjit, P., Austin, D.: Design of a modular self-reconfigurable robot. In: Australian Conference on Robotics and Automation. Citeseer, Sydney, Australia (2001)
17. Moreno, R., Gomez, J.: Simple chain type modular robot hardware (2011). https://www.youtube.com/watch?v=x6UQfC4KALA
18. Storn, R., Price, K.: Differential evolutiona simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)
19. Caamano, P., Tedin, R., Paz-Lopez, A., Becerra, J.A.: JEAF: A Java Evolutionary Algorithm Framework. In: IEEE Congress on Evolutionary Computation, CEC 2010, pp. 1–8, December 2007. IEEE, Barcelona, July 2010
20. Jakobi, N.: Evolutionary robotics and the radical envelope-of-noise hypothesis. Adapt. Behav. **6**(2), 325–368 (1997)